HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science And Engineering
Laboratory of Information Processing Science

Terje Bergström

Context Awareness in Symbian OS Based Smart Phones

Thesis is submitted for examination for the degree of Master of Science in Engineering

Supervisor: Professor Nisse Husberg

Inspector: Jere Seppälä, B.Sc.

TEKNILLINEN KORKEAKOULU
DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Terje Bergström	
Työn nimi:	**Kontekstitietoisuus Symbian OS -pohjaisessa päätelaitteessa**	
Päivämäärä:	19.8.2002	**Sivumäärä:** 50
Osasto:	Tietotekniikan osasto	
Professuuri:	Tik-76 Sulautetut järjestelmät	
Työn valvoja:	Professori Nisse Husberg	
Työn ohjaaja:	B.Sc. Jere Seppälä	

Tiivistelmä

Ihmisen kommunikointi toisen ihmisen kanssa sisältää paljon muutakin kuin pelkkiä sanoja. Suuri osa kommunikoinnin sanomasta perustuu siihen, että kommunikoinnin molemmat osapuolet tietävät jotain toisen historiasta tai kontekstista.

Ihmisen ja tietokoneen välisessä kommunikoinnissa kontekstia ei voi käyttää automaattisesti hyväksi. Tietokoneet eivät tarjoa palveluja, joiden avulla ohjelmistot voisivat muokata itseään käyttäjän kontekstin mukaan. Tämä ei ole suuri ongelma pöytäkoneissa, sillä niiden konteksti säilyy vakaana koko ajan.

Suuremmat ongelmat syntyvät kannettavissa laitteissa, kuten puhelimissa, joiden konteksti muuttuu jatkuvasti. Käyttäjän pitää syöttää tieto uudesta kontekstista aina, kun laitteen pitäisi reagoida siihen. Esimerkiksi kun käyttäjä siirtyy luentosaliin, hänen pitää laittaa puhelin hiljaiselle erikseen.

Tässä työssä tutkitaan mahdollisuutta rakentaa kannettavaan Symbian OS -pohjaiseen laitteeseen kontekstitietoisuutta. Tämä toteutetaan rakentamalla kontekstipalvelin, jolta ohjelmistot voivat pyytää tietoja käyttäjän kontekstista. Palvelimelle annetaan pääsy lämpötila-, valoisuus- ja äänianturiin ja tutkitaan, miten niiden antamaa tietoa voi käyttää kannettavan laitteen asettamissa rajoituksissa.

Avainsanat: konteksti, kontekstitietoisuus, itseorganisoituva kartta

HELSINKI UNIVERSITY OF TECHNOLOGY
ABSTRACT OF THE MASTER'S THESIS

Author:	Terje Bergström	
Name of the thesis:	**Context Awareness in Symbian OS Based Smart-phones**	
Date:	19.8.2002	**Number of Pages:** 50
Faculty:	Department of Computer Science And Engineering	
Professorship:	Tik-76 Embedded Systems	
Supervisor:	Professor Nisse Husberg	
Instructor:	Jere Seppälä, B.Sc.	

Abstract

The human communication includes a lot more than just the words. A major part of the communication is based on the fact that both parties to the communication know something about the history or context of each other.

In the communication between a human and a computer the context cannot be used automatically. The computers do not provide the applications methods for adjusting the user interface according to the user's context. This is not a great problem in desktop computers, where the context remains the same most of the time.

Greater problems occur in mobile devices, such as phones, whose context is constantly changing. The user must explicitly enter information about the new context every time the device has to react to the change. For example, when the user goes to a lecture hall, he must silence the phone explicitly.

In this research the possibility of building context-awareness into a Symbian OS based mobile device is studied. This is done by building Context Server, from which applications can request information about the user's context. The Context Server is given access to temperature, light and sound sensors and the usability of the data in the constraints of a mobile device is studied.

Keywords:	Context, Context-Awareness, Self-Organizing Map

Acknowledgments

I would like to thank the instructor, Technology Manager Jere Seppälä, for his advice and support, for letting me work on this thesis, and for helping me present my ideas to the company management. In addition, I would like to thank the supervisor, Professor Nisse Husberg, for giving me advice in scientific writing. I would like to thank the English teacher and senior lecturer Elizabeth Heap-Talvela for helping me with my English grammar.

Many thanks go to all my colleagues in Digia for creating a supportive and relaxed atmosphere, and supporting me with my work. Without the flexible culture and work hours of Digia, I would never have been able to finish my studies.

Special thanks to my wife for supporting me when I was tired and enduring me when I spent my days working and studying, and did not have enough time for her. Another special thanks to my father, mother and sister for supporting me in my computer hobby, and creating the atmosphere, where it was easy to make a career out of it.

Terje Bergström

Contents

List of Figures

List of Tables

List of Abbreviations

FSD	Full Scale Deflection
GSM	Global System for Mobile Communications
KSOM	Kohonen's Self-Organizing Map
MVC	Model-View-Controller
OS	Operating System
PC	Personal Computer
RFID	Radio Frequency Identification
UML	Unified Modeling Language
USB	Universal Serial Bus
YTV	Pääkaupunkiseudun yhteistyövaltuuskunta. Helsinki Metropolitan Area Council.

Chapter 1

Introduction

1.1 Background

Computers have traditionally been considered stupid and unable to understand their users. Despite their superior computational power, the computers have not been able to reach the level of communication that humans have between each other. A computer is insensitive to the implicit methods of human communication — gestures, tones of voice and the context of the communication.

This research focuses on one of the implicit communication methods: the context. When a human talks to another human, they share parts of their context, for example the location or history. They can take advantage of this context while interpreting what the others are saying. Thus, the total bandwidth of their communication is very high.

When people communicate with computers, they have to do it via a keyboard or a similar input device. They have to consciously remove all context information from the message before entering it in computer, or embed the context information explicitly in the commands. If a computer has to react to changes in the environment, the users have to tell the computer explicitly how it should react. The computer cannot easily access the situational data of the user, even though some parts of it could be retrieved automatically. Part of the contextual information can be deduced from the data the computer already has, but a much richer pool of context data can be accessed. This can be achieved by giving the computer access to a range of cheap sensors, which sample the surrounding environment.

Normal desktop computers remain most of the time in a relatively static context

and thus the lack of context-awareness is not considered a big problem. The problem becomes much worse with mobile devices, which the users carry around everywhere they go. The context of the devices changes continuously and the users have to adjust the device explicitly to the new situation. For example, if a user enters a lecture hall, he[1] has to silence the mobile device. If he walks outside with a thick coat, he has to increase the volume of the ring tone so that he can hear the call. Sometimes this adjustment can be difficult, or even dangerous, if the situation requires the user's attention.

1.2 What is Context and Context-Awareness?

In this research, a very broad definition of context proposed by Dey and Abowd is used[4].

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

This definition includes implicit context, such as information about the surrounding environment of the user, but also context explicitly given by the user, such as choosing the current profile on a mobile phone or launching an application. They can both in part be used for determining the situation of the user. It also includes contextual hints given by external objects, such as radio frequency identification badges when moving inside a building. It excludes all information not relevant to the communication between applications and users.

This research focuses only on a small subset of context types. They are location, identity, time and activity. According to Dey, they can be used for deriving further context information [3]. For example, when the identity, location and time of the user are known, a calendar application can check if the user is on his way to a meeting or if he should be reminded about it.

In defining the term context-awareness, again a definition proposed by Dey and Abowd is used[4].

[1] The user is referred in this research with the masculine pronoun he, even though the user can as well be female

> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

This definition is based on the relevancy of information to the user. A system does not have to act or modify its behavior according to the context, if it is deemed not necessary. It also does not explicitly state how a system should react to the context. It only states that a system should give its user some information or services relevant to the context.

In the same publication, Dey and Abowd provided a categorization for context-awareness [4]. They proposed three categories: presentation of information and services to a user, automatic execution of a service, and tagging of context to information for later retrieval. These are all considered in this research, and example applications are described for each type of context-awareness.

1.3 Research Problem

This research deals with the following questions:

1. How can the context information be acquired?
2. What kind of contextual information is a mobile device able to use?
3. What kinds of sensors are needed to produce meaningful information?
4. How could applications take advantage of the context information?

These questions are discussed in the theoretical point of view, but to validate the answer, it has to be tried in practice. An experimental software system Context Server was developed to validate the answer.

1.4 Research Goals

The first goal was to determine what kind of sensor information a mobile device is able to process to produce meaningful information about its context. The goal

was to choose algorithms that are light on computational resources, but still can produce useful information.

The second goal was to create a system, Context Server, which enables applications to take advantage of the context easily. The system is able to retrieve information about the surroundings of a device via sensors and produce contextual information, which can be used by applications without complex processing. Applications in a mobile device can connect to the Context Server to acquire contextual information and to receive notifications about changes in the context.

The third goal was to produce a small example application, which is able to take advantage of the contextual information created by Context Server. The application together with Context Server constitutes the software prototype.

The last goal was to assemble a hardware prototype, which consists of sensors, media to transfer the sensor data to a mobile device, and the mobile device. The software prototype is demonstrated in this hardware prototype.

As stated earlier, this research focuses only on a subset of context types. This is due to the limited computational power provided by a mobile device, but also due to lack of research in acquiring other types of context.

Although signal-processing algorithms must be applied to the received sensor data, they are not the topic of this research. Example algorithms are chosen, applied, and described briefly, but they are not studied more deeply.

In the prototype, several novel technologies were applied, such as Bluetooth and a Symbian OS based smart-phone was used as the platform. Nevertheless, they are not an essential part of the research, but merely tools to accomplish the goals. Thus, only the parts that are relevant to obtaining the results are described.

1.5 Organization of the Thesis

Chapter 2 is an overview of the previous work done in the area. It also describes all the algorithms used, such as self-organizing map and k-means clustering. Chapter 3 introduces all the tools and devices used for realizing this research. Chapter 4 describes the Context Server, the related example application and measurements of performance. Chapter 5 is an analysis of the results described in Chapter 4. Finally, Chapter 6 presents conclusions made in this research.

Chapter 2

Background and Related Work

2.1 The Context Toolkit

The Context Toolkit was a framework developed by Dey [3]. The framework was designed to make it easy for application designers to take advantage of situational information. The contribution of Dey was identifying the need for a framework. He also described a design process for building context-aware applications and pointed out the parts of the development process that can be avoided. He identified the following steps in the process.

1. *Specification*: Specify the problem being addressed and suggest a high-level solution
2. *Acquisition*: Determine what new hardware or sensors are needed to provide that context.
3. *Delivery*: Provide methods to support the delivery of context to one or more, possibly remote, applications.
4. *Reception*: Acquire and work with the context.
5. *Action*: If context is useful, perform context-aware behavior.

He classified these steps into essential and accidental activities. Essential activities are the ones that really must be carried out, because they are specific to an application. Accidental activities are necessary only because there is no support

in the system to take care of them. He also described the revised design process, where all accidental activities had been removed.

1. *Specification*: Specify the problem being addressed and a high-level solution.
2. *Acquisition*: Determine what hardware or sensors are available to provide the context and install them. This is not needed if the necessary sensors are already in place.
3. *Action*: Choose and perform context-aware behavior.

Dey proposed that by implementing proper abstractions and services for handling contexts, it is possible to build a framework that makes it easy for application programmers to take advantage of context information. This was the major inspiration for this research.

He went on to describe basic framework features that were required in a context framework.

Context specification is a mechanism that allows application programmers to specify the required context types

Separation of concerns and context handling means splitting the mechanisms for retrieving the context from how it is used in applications

Context interpretation means interpreting sensor data to provide abstract context information to the applications so that the applications do not need to interpret the information any further.

Transparent distributed communications allows hiding information on the physical location of the sensor or applications.

Constant availability of context acquisition separates the components that retrieve context from the components that use them

Context storage is a mechanism for storing historical information about the context

Resource discovery is a way of finding certain types of sensors or context components automatically

These features are relevant to a fully distributed environment, where the sensors can be located anywhere, and the software components run on powerful desk-

top computers. This research focuses only on an environment, which is run on a low-power mobile device with minimal support from hardware. Thus, the system described in this research will focus on only a subset of these features, namely Context specification, Context interpretation, Constant availability of context acquisition and Resource discovery.

2.2 Technology for Enabling Awareness

The Technology for Enabling Access project was an attempt to provide context-awareness to mobile devices [13]. The prototype developed used a sensor-board with a microcontroller connected to a PC to acquire and process the sensor data. An ordinary Nokia 6110 phone was connected to the PC to take advantage of the context information provided by the PC.

The software architecture was divided into layers. The outputs of the lower layers are used in the higher layers as input. The first layer consists of the physical and logical sensors. Physical sensors are electronic hardware components. Logical sensors acquire data from the host, for example the current time and GSM cell. The second layer consists of cues, which are an abstraction of the sensor data. A cue is a function that takes input from the physical or logical sensors and provides abstracted information as output. In the third layer are actual abstract contexts, which describe the current situation. The contexts are derived from cues. The applications occupy the fourth layer.

The aim of this research is to create a system where as much processing as possible is done in the mobile device. This way the sensors can be cheap and the system is less dependent on certain types of sensors.

2.3 Feature Extraction

The input data gathered in a defined period from the sensors can be considered a vector to be fed into the pattern recognition algorithms. This presents great problems, because the dimensionality of the vector is high, and the memory requirements and complexity of the processing would be well above the abilities of a mobile device.

The raw input data contains a lot of noise and repeating, superfluous data. There-

fore, algorithms are used to extract only the relevant information from the raw data[9]. This extraction is called feature extraction, because it takes the raw signal data and tries to characterize it in a small number of variables — in other words, it takes the raw data, tries to find its characteristic features, and outputs them as feature vectors.

The simplest algorithm for feature extraction is to calculate a statistical property for a signal in a specified period. In the Technology for Enabling Awareness project the following properties were used [13]: average, standard deviation, quartile distance, base frequency, and first derivative.

Gerhard in his report suggested the following six properties [6].

Energy can be used for detecting silence from noise. More advanced algorithms calculate energy in different frequency bands.

Zero-crossing rate is the number of times a signal crosses the zero-level per unit time. This measurement gives information about the spectral content of the signal. It is advantageous to use a low-pass filter as a pre-filter.

Spectral Features are calculated from the spectrum of a signal. They quantify the energy distribution of a signal to different frequencies. These algorithms need significant processing power, so they are not considered here.

Fundamental Frequency can be calculated for periodic signals. It is the lowest frequency of an instrument, and is often called the first harmonic.

Formant Location is used for describing the peak frequencies of human speech. This can mainly be used for voice recognition and differentiating different speakers.

Time-Domain Features and *Modulation* do not need much processing power, but usually they provide only little information. For example, the duration of a signal can be used for separating a finger snap from speech.

Those algorithms that need the computation of a Fourier transform cannot be used. This includes all *Spectral features* algorithms, but also the calculation of energy in different frequency bands. This is because even the fastest Fourier transform algorithms need more processing power than the processor mobile device can provide. In fact, they could be applied if the digital signal processor of the mobile device could be accessed. Unfortunately, this is not the case in any mobile device in the market.

2.4 Clustering of Data

Even though feature extraction makes the dimensionality of input data lower, the feature vectors still are large, and their range of values can be enormous. The feature vectors can be reduced even further with clustering[9]. It classifies the feature vectors into a smaller number of subsets. Each subset is described with a vector, which approximates all the input vectors that have been mapped to the subset. These are called reference vectors in this research.

2.4.1 Kohonen's Self-Organizing Map

Kohonen's self-organizing map is an adaptive algorithm for clustering input data [10]. The input vectors $\vec{x}$ can be of any dimension. The map stores reference vectors $\vec{m}_i$, which are of the same dimensionality as the input vectors, in the nodes of a two-dimensional grid.

For every input vector $\vec{x}$ provided to the network, the best-matching node c is determined. This matching can be done with any metric, but in many applications, the Euclidian distance is used:

$$\|\vec{x} - \vec{m}_c\| = \min_i \{\|\vec{x} - \vec{m}_i\|\}. \tag{2.1}$$

This is called the Competitive Process of the algorithm [7].

The reference vector of the winning node is moved closer to the input vector, thus making it more sensitive to the particular vector. This is called the Cooperative Process [7]. Topological mapping is achieved by moving the neighbors of the winning reference vector towards the input vector as well, but not as much as the winning neuron. The formula for updating all the reference vectors in the map is

$$\vec{m}_i(t+1) = \vec{m}_i(t) + h_{ci}(t)[\vec{x} - \vec{m}_i(t)], \tag{2.2}$$

where $h_{ci}(t)$ is the neighborhood function centered on the winning neuron. It should decrease with time to ensure convergence. An example function is a Gaussian function:

$$h_{ci}(t) = \alpha(t) \exp\left(-\frac{\|\vec{r}_c - \vec{r}_i\|^2}{2\sigma^2(t)}\right). \tag{2.3}$$

The parameter $\alpha(t)$ is the learning-rate factor. For approximately the first 1000 steps, so-called ordering phase, it should decrease monotonically. After the ordering phase, it should assume a small value to enable stable operation in the operational phase.

The function $\sigma(t)$ controls the radius of the neighborhood. It should decrease monotonically as the value of t increases. This decreases the likelihood that reference vectors are overwritten with input from other classes.

Initially the reference vectors can be set to random vectors. In the initial training of a self-organizing map, the map goes through a so-called ordering phase. In that phase the reference vectors move in the map so that the topological ordering is established. In the second, so called convergence phase, the statistic characteristics of the input vector space are incorporated into the map.

2.4.2 K-Means Clustering

K-means clustering is similar to the self-organizing map[8]. The number of clusters l is decided when the algorithm is initialized. Each cluster consists of input vectors, and for each cluster, the centroid of all its input vectors, $\vec{m}_i$ is maintained.

When an input vector $\vec{x}$ is fed to the algorithm, for each cluster the distance from $\vec{x}$ to $\vec{m}_i$ is calculated with Euclidian algorithm

$$d_i = \|\vec{x} - \vec{m}_i\|, i \in [1 \ldots l]. \tag{2.4}$$

The input vector is added to the cluster c with the centroid closest to the input vector. After adding, the centroid $\vec{m}_c$ of the cluster is recalculated. The algorithm is initialized by setting the $\vec{m}_i$ for all clusters to a random value.

The problem with k-means clustering is that all input vectors need to be stored and kept during the lifetime of the algorithm. This can be corrected by replacing the calculation of $\vec{m}_c$ with an estimating algorithm [17]

$$\vec{m}_c(t) = \vec{m}_c(t-1) + \lambda \cdot (\vec{x} - \vec{m}_c(t-1)), \tag{2.5}$$

where λ is a constant learning-rate parameter.

The main difference between a self-organizing map and the estimating k-means clustering is that in k-means clustering only the winning cluster is adjusted. In the self-organizing map, also the neighbors of the winner are adjusted.

2.4.3 Adding K-Means Clustering to Self-Organizing Map

A self-organizing map in itself does not provide labeling. This means that the map approximates the input space, but does not allow classification of input vectors. Classification is essential for this research, because the input vectors need to be mapped to the corresponding context types. Given an input vector, the map should be able to find the corresponding context type, of which the applications can take advantage.

Another problem with Kohonen's self-organizing map is that the learning continues all the time. This can lead to overwriting previously learned data, especially when the map is in ordering phase [17]. A neuron, whose reference vector belongs to one class, can have its reference vector overwritten by a vector from another class.

van Laerhoven proposed augmenting the Kohonen's self-organizing map with a second layer for each label, which does on-line k-means clustering of labeled input vectors. This means that after determining the winning neuron for an input vector, the reference vector of the neuron is fed to the k-means clustering algorithm.

If the input vector $\vec{x}$ is labeled, it will be clustered to the k-means layer of the label j:

$$\vec{m}_{jl} = \vec{m}_{jl}(t-1) + \lambda \cdot \left(\vec{x}(t) - \vec{m}_{jl}\right), l \in [1,k]. \tag{2.6}$$

Unlabeled data is classified by choosing the k-means layer with the cluster whose centroid $\vec{m}_c$ is closest to the input vector.

van Laerhoven also proposed that the learning-rate factor $\alpha(t)$ and width of the neighborhood function $\sigma(t)$ in the self-organizing map should be functions of time spent in the current context, instead of functions of overall time used.

2.5 Symbian OS

Symbian OS is a robust, multitasking 32-bit object-oriented operating system [15][16]. It can be run on several hardware configurations, for example Sony Ericsson P800 and Nokia 7650. It provides the programmer with a coherent and comprehensive set of programming interfaces and an event based programming model, which make it simple to write efficient and portable applications.

Some aspects of Symbian OS differ from the way other operating systems work. Therefore, the main technologies that enable creating the Context Server are introduced here. The other technologies provided by Symbian OS are ignored here.

2.5.1 Dynamic Link Library

A dynamic link library is a module, which can be loaded into memory when needed. It provides applications with a predefined set of functions, which constitute a programming interface. The author of the library can decide which functions he wants to keep internal and which functions can be called by applications. Making a function available for applications is called exporting. Each exported function is allocated an entry point in the dynamic link library. The entry points are stored as an array of pointers in the beginning of the library.

A dynamic link library is loaded into memory once, and after that, all applications that use it share the same copy of the library. Because the functions are called via entry points, the application does not need to know the physical location of the function in address space.

2.5.2 Client-Server Framework

In Symbian OS, a server is an application with no user interface [16]. A server has a message port, through which clients can request services[14]. Before requesting services, the client has to register itself to the server by opening a session. The session acts as a holder of state information when communicating between the server and the client.

The author of the server generally also writes a programming interface, which takes care of the communication between an application and a server. The programming interface is packaged as a dynamic link library. In many cases, the user

of the dynamic link library does not need to know that a server is used for performing the tasks.

The Advantages of Client-Server Framework

Servers are usually used when there is a resource, which the applications should share with no conflicts. Because servers in Symbian OS are written using an event-based programming model, they are generally single-threaded and thus easily enforce consistency of the resource.

Another case where servers are needed is a service, which must be running all the time, independently of the applications. For example, the Messaging Server must be running always so that incoming faxes or short messages can be retrieved by the mobile device regardless of the applications open at the time.

A third reason for using servers is efficiency. Servers can act as caches for commonly accessed information.

The last, but not the least, reason for using servers is information consistency and reliability. Servers live in their own processes and address spaces, which isolates them from bugs in the client software.

Messaging Between Client and Server

Messages sent by the client consist of the session number and five 32-bit numbers. The first number specifies the command. It can be any integer as long as the client and the server agree on it. The remaining four integers are parameters to the command. The server can respond to the message with a single integer. It is used to report the success or the failure of the command.

Five 32-bit numbers is very limiting, if the size of the data to be transferred is large. The parameters of a message can also specify an address in the client's address space. The server can read larger blocks of data directly from the client's address space or write data by using services provided by the kernel.

Telephony Server

Symbian OS is designed to work in telephone-like equipment. Thus, it is essential for the applications to be able to use the services of the phone to, for example,

send a short message or a fax.

The Telephony Server acts as an intermediary between the application and the phone hardware. It can be used for accessing all the services that a GSM network can provide to the phone ranging from placing voice calls to retrieving the cell identifier of the currently serving GSM cell.

Agenda Server

The calendar of the user is a resource, which can be needed by multiple applications at once. Examples are the normal Calendar application, alarm clock and the PC synchronizer, which synchronizes the calendars on the PC and mobile device. These situations could create conflicts, if the calendar information is stored in a single file, and all applications access it directly.

The situation is solved in Symbian OS with an Agenda Server. It takes care of all file manipulation, and provides the applications with a rich set of programming interfaces to read from or write to the calendar. Because a calendar can contain images and other objects, the calendar file can grow large. The Agenda Server acts as a cache to speed up operations for applications.

2.5.3 Server Plugins

A server is built as a single executable file onto the target platform. This does not allow writing extensions to the server. This is a major restriction in many cases, because the environment of a mobile device is dynamic and often cannot be predicted by the author of the server. To overcome this problem, Symbian OS provides polymorphic dynamic link libraries. For the sake of shortness, they are called plugins in this research.

Plugins are like normal dynamic link libraries, except that they only contain one entry point. This entry point points to a function that usually constructs an object with a predefined interface. After the object is constructed, the server can call the methods of the object directly.

Plugins are used heavily in Symbian OS[14]. They are used for loading device drivers, communications modules and even file systems.

Chapter 3

Material and Research Methods

3.1 Building Blocks of the Prototype

3.1.1 The Sensors Providing Environmental Data

Most of the sensor data was acquired with DrDAQ[5]. It is a data logger from Pico Technology, which contains built-in sensors for temperature, light and sound. It also contains sockets for external sensors and can measure resistance and voltage. The specifications of the data logger are described in Table 3.1.

The DrDAQ data logger plugs into the parallel port of a PC. It comes with drivers for Windows 2000.

Type	*Range*	*Resolution*	*Accuracy*
Sound waveform	±100	0.2	Not calibrated
Sound level	55 – 100 dBA	1 dBA	5 dBA
Resistance	0 – 1 MΩ	0.1 kΩ (at 10 kΩ)	2% (at 100 kΩ)
Voltage	0 – 5 V	5 mV	3% of FSD
Temperature	0 – 70° C	0.1° C (at 25° C)	2° C (at 25° C)
Light	0 – 100	0.1	Not calibrated

Table 3.1: DrDAQ data logger technical specifications

	Desktop	Laptop
Make	IBM	IBM
Model	Intellistation E Pro	ThinkPad T20
Processor	Pentium III 600MHz	Pentium III 750MHz
Hard disk	10GB + 10GB	20GB
Memory	384MB	256MB
I/O Ports	2× USB, Parallel, 2× Serial	1× USB, Parallel, 1× Serial

Table 3.2: Specifications of the computers used as data logger

3.1.2 Router between Sensors and Communication Media

A desktop computer was used for reading the data from sensors and transmitting it to the mobile device via a radio path. The desktop computer does not process the data in any way. Also using a laptop was attempted, but the DrDAQ data logger did not work with it.

The computers used are described in Table 3.2. The most important ports are the serial and parallel ports and USB.

A special program was written that periodically reads data from the DrDAQ data logger and writes it into a file. Because Ericsson's Windows Bluetooth stack did not work as expected, another program was created to run in the emulator of Symbian Quartz 6.1 software development kit, which contains its own Bluetooth stack. The program reads the DrDAQ sensor data from a file and sends it in packets to the mobile device.

The development tools used were Microsoft Visual C++ 6.0, Ericsson Bluetooth Application Tool Kit drivers, DrDAQ drivers, Nokia Series 60 SDK for Symbian OS and Symbian Quartz 6.1 Software Development Kit.

3.1.3 Communication Media for Transmitting Environmental Data

The Ericsson Bluetooth Application Tool Kit was used for sending the sensor data to the mobile device[1]. It connects to a PC via USB or serial bus, and comes with drivers and development tools for Windows 2000. The Symbian Quartz 6.1 software development kit supports the Ericsson Bluetooth Application Tool Kit directly. In addition, the mobile device used supports Bluetooth.

Thus, Bluetooth is used as the interface between the mobile device and desktop computer. The device is connected to the desktop computer via the serial port, because the Bluetooth stack used does not support USB. The device is connected also via the USB, because the device draws power from the USB port of the computer.

3.1.4 Mobile Device Running the Context Server

The Nokia 7650 imaging phone was used as the mobile device[12]. The phone runs the Symbian OS 6.1 operating system and is an open platform. That means that applications can be created by using the software development kit from Nokia and installed onto the phone. It contains built-in Bluetooth connectivity.

The Context Server runs on the phone and waits for the sensor data. After receiving the data, it is processed and entered into the Context Server. In addition to the sensor data, the Context Server is able to retrieve some information directly from the phone using the services provided by the operating system.

As in every Symbian OS platform, also the Nokia 7650 Software Development Kit comes with a software emulator. It allows creating and testing software on a desktop computer with the debugging facilities of Visual C++. The same source code of software can then be compiled for the target platform and run on a mobile device.

3.2 Data Acquisition

Data acquisition is the process of collecting data from various sources. In this research, the data sources used include physical sensors, such as the DrDAQ data logger, but also logical sensors.

3.2.1 Input Signals from DrDAQ

The input signals from the temperature, light and sound sensors were processed before entering them into the Context Server. The following properties were calculated from the raw input data: minimum value, maximum value, average value, and zero-crossing rate. Please see Chapter 2.3 for the explanation on the properties. They were chosen, because they are fast to calculate, but still should be

usable for differentiating between contexts. These four values were entered into the Context Server to be processed further with other algorithms.

3.2.2 Location Based on GSM Positioning

The mobile device can always get some parameters from the network, which can be used for approximating its location. The known parameters are country code, network code, location area, and cell identifier. The country code is the country where the network is located. The network code identifies the operator within the country. The location area identifies a group of cells of a single operator and the cell identifier identifies a single GSM cell uniquely. The Telephony Server provides this information to the applications.

Location is in this research defined as a geographical area, which has a meaning to the user. It can be a logical name, such as home and work, but also a named place such as a street name or a bus stop. The size of the area varies, but it cannot be smaller than a single GSM cell.

The network information alone cannot be used for determining location, but the Context Server can be taught to map the network parameters to locations. This is done by asking the user to label the location when a new network parameter is acquired. The next time the same network parameters are received, the Context Server can look up the location from its database.

3.2.3 Using Calendar for Analyzing User's Intent

The Nokia 7650 contains a calendar application. Each note in a calendar can contain the subject, location, start time and duration of a meeting. The interesting information is the location, which can be mapped to the location derived from the network parameters, and the start and end times of the meeting. That information can be used for determining where the user desires to be at a certain point of time.

The calendar was periodically searched for the next meeting from the current time. Its information was entered to the Context Server. The calendar was accessed via the Agenda Server, which is described in Chapter 2.5.2.

Attribute
Average temperature
Average light level
Average sound level
Average sound waveform
Sound waveform minimum
Sound waveform maximum
Sound waveform zero-crossing rate

Table 3.3: The statistical information from sensor data that is entered into the Context Server.

3.2.4 Clustering and Classification of Environmental Data

The data entered into the Context Server, listed in Table 3.3, was used for deriving a higher-level description of the user's context. The attributes were collected into a vector, which was entered into an augmented Kohonen's self-organizing map described in Chapter 2.4.3.

Because the Nokia 7650 does not contain a floating-point unit, which makes the floating-point operations very slow, the neighbor function was chosen so that floating point mathematics are not needed. The Equation 2.3 was replaced with the following linear function:

$$h_{ci}(t') = \alpha(t') \cdot \left(\frac{2 - 2t'/101}{\|\vec{r}_c - \vec{r}_i\| + 1} + 1 \right), t' \in [0, 100], \tag{3.1}$$

where t' is the time spent in current context. The learning-parameter α was calculated as a linear function of t':

$$\alpha(t') = \frac{1 - t'/101}{20}, t' \in [0, 100]. \tag{3.2}$$

The k-means clustering layer consists of multiple clusters per label. The algorithm checks how close the vector is to an existing cluster. If the distance is small, the vector is fed to the cluster. Otherwise, a new cluster is created.

The update function for the cluster centroid $\vec{m}_c$ was adjusted to ensure that the algorithm does not forget the oldest input vectors. The algorithm keeps track of how many vectors there are in each cluster in variable n. The algorithm

$$\vec{m}_c(t) = \vec{m}_c(t-1) + \frac{1}{n} \cdot (\vec{x} - \vec{m}_c(t-1)) \tag{3.3}$$

was applied in the system.

3.3 Data Source and Research Methods

The locations were taught to the system by taking the mobile device to different locations and entering the locations into the database. This was done by carrying the mobile device around without the computer or DrDAQ data logger.

Test data was gathered by exposing the measurement equipment to different contexts. Because the DrDAQ data logger did not work in the laptop used, the desktop computer had to be used for data gathering. Due to this, the range of context types was limited to context types in an office environment. The data was collected, and labeled and entered offline into the Context Server.

The research was constructive. In other words, the research was conducted by creating the Context Server. Its performance was evaluated by testing how a mobile device is able to run the Context Server and its algorithms. The accuracy of algorithms was measured in an emulator environment on a desktop. The processing time efficiency was measured in the mobile device.

In addition, an example application was created to demonstrate the feasibility of context-awareness in today's mobile devices.

Chapter 4

Results

4.1 Context Server Actors and Use Cases

The designing of the Context Server was started by identifying the use cases it has to support in the spirit of Unified Modeling Language[2]. The use cases are presented here in an informal textual form, and as use case diagrams.

4.1.1 Actors

There are four actors in the Context Server system.

Client is an application, which connects to *Server*, and requests services from *Server*.

Observer is a module, which is interested in changes in the context information. The types of contexts the module is interested in can be specified. *Observer* is a specialization of *Client*.

Provider is a module, which produces context information. *Provider* is a specialization of *Client*.

Server is the module, which satisfies requests from *Clients*. It keeps a database of context information.

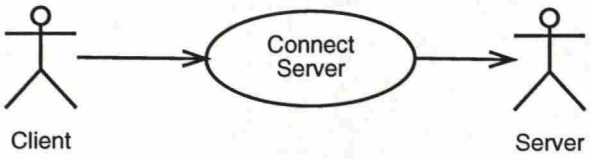

Figure 4.1: UML diagram of the use case Connect Server

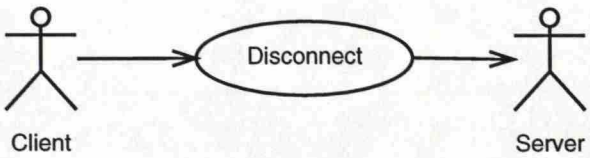

Figure 4.2: UML diagram of the use case Disconnect

4.1.2 Connect Server

Client needs to connect to *Server* before it can use *Server*. If *Server* is not running, it has to be started before connecting. After connecting there exists a session between *Client* and *Server*. See Figure 4.1 for a UML diagram.

4.1.3 Disconnect

When the user exits an application, *Client* has to disconnect from *Server*. This will free any resources reserved for the session and cancel any outstanding request. See Figure 4.2 for a UML diagram.

4.1.4 Retrieve Context

Client needs a certain type of context information. It sends *Server* a request for the information. If *Server* has the particular information, it will be sent to *Client*. Otherwise, an error is reported. See Figure 4.3 for a UML diagram.

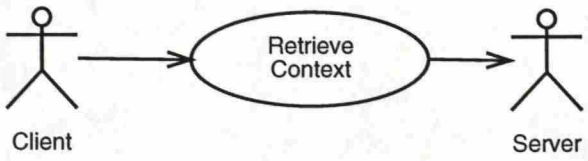

Figure 4.3: UML diagram of the use case Retrieve Context

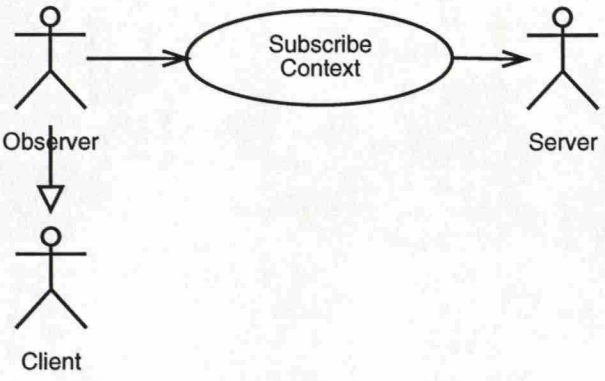

Figure 4.4: UML diagram of the use case Subscribe Context

4.1.5 Subscribe Context

Observer wants to be notified when a certain type of context information is modified. It sends *Server* a list of context types it is interested in, and asks *Server* to notify *Observer*. *Server* stores the information. See Figure 4.4 for a UML diagram.

4.1.6 Provide Context

Provider has new information on a context type. It sends the information to *Server*. *Server* makes a copy of the information and stores the copy in its database. If there was already information on the context type, the old information is overwritten. Use case Notify Observer is invoked. See Figure 4.5 for a UML diagram.

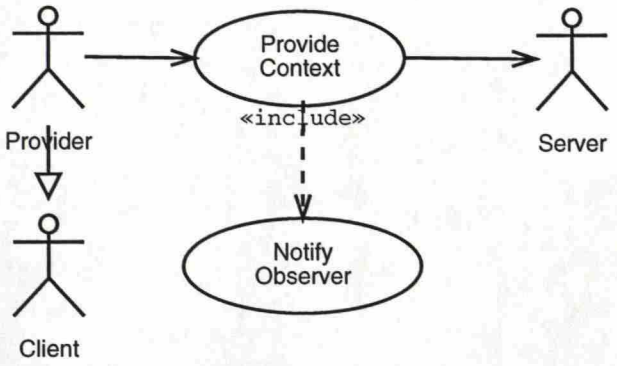

Figure 4.5: UML diagram of the use case Provide Context

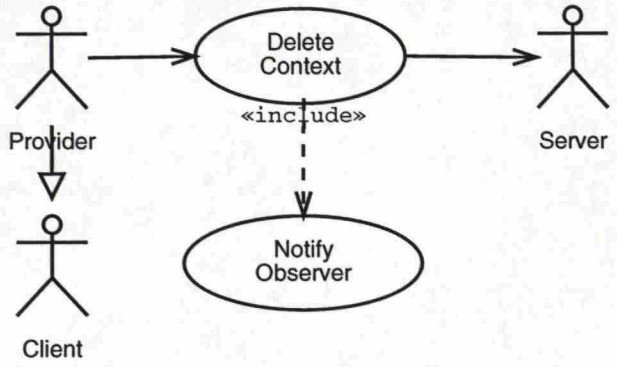

Figure 4.6: UML diagram of the use case Delete Context

4.1.7 Delete Context

A piece of context information is no longer valid, and new information has not been received. *Provider* wants to remove the old data from *Server* so that other *Clients* will not act on expired information. *Provider* sends a request to *Server* to delete the information. If the information exists in *Server's* database, it is deleted. Use case Notify Observer is invoked. See Figure 4.6 for a UML diagram.

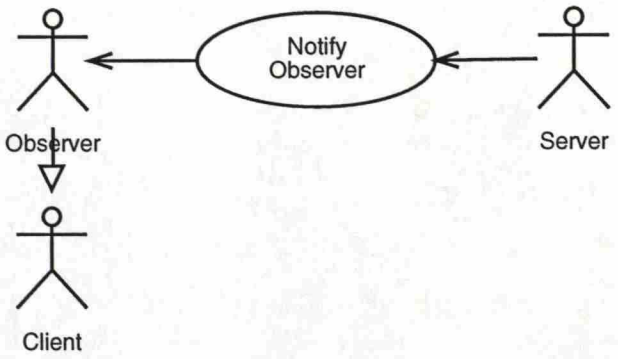

Figure 4.7: UML diagram of the use case Notify Observer

4.1.8 Notify Observer

Some context information has been changed. All *Observers* are scanned, and the ones that are interested in the particular type of context information are notified. *Provider* of the particular piece of information is not notified to prevent infinite recursion. See Figure 4.7 for a UML diagram.

4.2 The Context Server System

The Context Server consists of three modules: the Context Server itself, the Context Engine, and a client interface. In addition to this, modules can be loaded as plugins.

In Symbian OS, there are certain conventions for using C++[11]. The conventions were obeyed in naming the classes and methods of the Context Server, so they are described here.

The normal classes, which do not allocate memory from the heap, are called T classes. Their name starts with the letter T. Classes, which allocate memory from the heap, must be derived from class `CBase`, and their names start with the letter C. Interface classes consist only of virtual abstract methods. They must not contain any implementation. Their names start with the letter M. All classes have also the identifier `Cx` to prevent name collisions with other software.

Methods can use the Symbian OS exception system, which is called leaving.

Methods, which can leave, have the letter L appended to them.

In C++, multiple inheritance is allowed, but in Symbian OS, its use is forbidden except in one case. When a class derives from a normal class and one or more interface classes, multiple inheritance is allowed.

The methods for constructing and destructing are presented in the method lists as single entries called construction and destruction. This is because presenting them would not convey any further information, because they are the same for all classes.

4.2.1 Context Engine

The Context Engine contains three classes: `CCxContextBoard`, `CCxContext-Notifier` and `CCxContext`. In addition to the classes, the Context Engine defines an interface `MCxContextObserver`.

Each `CCxContext` object describes one piece of information about the context surrounding the user. For example, the location of the user is described as a `CCxContext` object. Each piece of context information is described as a context name, which is an integer, and a value, which is a sequence of binary data. The context name describes, which piece of information is described in the particular `CCxContext` object. The context value contains the actual information, but it is up to the user of the `CCxContext` object to interpret its contents. The methods provided by `CCxContext` are described in Table 4.1.

Method	Description
construction	Constructs a `CCxContext` with a given name
destruction	Deletes an instance of `CCxContext`
SetValueL	Deletes the old value of the `CCxContext` instance and sets a new one
Value	Returns a reference to the value of `CCxContext`
Name	Returns the context name of `CCxContext`

Table 4.1: Methods provided by class `CCxContext`

`CCxContextNotifier` keeps a list of observers. Any object with access to an instance of `CCxContextNotifier` can add itself as an observer for certain kinds of context information by implementing the interface `MCxContextObserver` and registering to the `CCxContextNotifier`. Whenever any context information is

changed, the class CCxContextNotifier goes through all the observers and calls the ones who are interested in the particular kind of information. The methods provided by CCxContextNotifier and MCxContextObserver are described in Tables 4.2 and 4.3.

Method	Description
construction	Constructs a CCxContextNotifier
destruction	Deletes an instance of CCxContextNotifer
AddObserverL	Adds a observer for the specified context names
RemoveObserver	Removes an observer
NotifyL	Notifies all observers of a given context name

Table 4.2: Methods provided by class CCxContextNotifier

Method	Description
ContextChangedL	Called when a context of a subscribed type has been given a new value

Table 4.3: Methods provided by interface MCxContextObserver

CCxContextBoard is the owner of all CCxContext objects. All modifications of context data are done via the CCxContextBoard. It starts as empty, and information is added to it as it is produced. After each modification of context data, CCxContextBoard calls the CCxContextNotifier to send change notifications. The CCxContextBoard also provides methods for adding or removing observers from the CCxContextNotifier. The methods provided by CCxContextBoard are described in Table 4.4.

Figure 4.8 shows a UML diagram of the relationships of the Context Engine classes.

4.2.2 The Context Server

The Context Server is a single executable file. It consists of the core, classes CCxServer and CCxSession, and a plugin loader mechanism, which consists of classes CCxContextLoader and CCxContextDll. Each plugin must implement the interface MCxContextProvider.

The CCxContextLoader is used for loading server plugins. It scans for plugins in a designated folder. For each plugin found, an instance of CCxContextDll is

Method	Description
construction	Constructs a `CCxContextBoard`
destruction	Deletes an instance of `CCxContextBoard`
AddContextL	Adds a `CCxContext` to the board
ModifyContextL	Gives a context a new value
DeleteContextL	Deletes a `CCxContext` from the board
ContextL	Returns a `CCxContext` with a certain context name
Contexts	Returns a list of context names
SetNotifierL	Sets a `CCxContextNotifier` for this board
AddObserverL	Sets an observer for a certain context type to the `CCxContextNotifier`
RemoveObserver	Removes an observer from `CCxContextNotifier`

Table 4.4: Methods provided by class `CCxContextBoard`

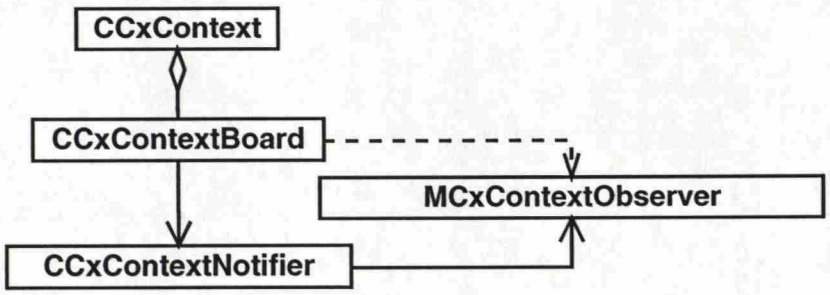

Figure 4.8: UML diagram of the Context Engine

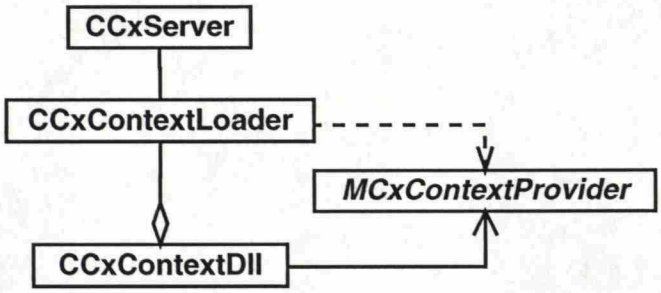

Figure 4.9: UML diagram of the Loader of the Context Server

created. It loads the plugin and calls its entry point. The plugin must return an object, which implements the interface `MCxContextProvider`. Often the plugins also implement the interface `MCxContextObserver` to receive notifications about new context information. Figure 4.9 contains a UML diagram of the loader. The methods of the interface `MCxContextProvider` are described in Table 4.5.

Method	Description
ActivateL	Activates the provider by giving it a pointer to the `CCxContextBoard`

Table 4.5: Methods provided by interface `MCxContextProvider`

The core, an instance of `CCxServer`, uses the Context Engine to keep track of context information. It allocates a single instance of the `CCxContextBoard`, and `CCxContextNotifier`, and signals the operating system that the server is functioning. After that, clients can connect to the server. For each client, an instance of `CCxSession` is created. It handles all interaction between the server and the client from the viewpoint of the server. Figure 4.10 contains a UML diagram of the core.

The methods of the classes in the core are not described further, because they are deeply connected to the framework of Symbian OS and not relevant to this research.

4.2.3 Context Client

The client is a dynamic link library. To use it, an application links to the library, and creates an instance of `RCx`. `RCx` implements methods, which allow the applica-

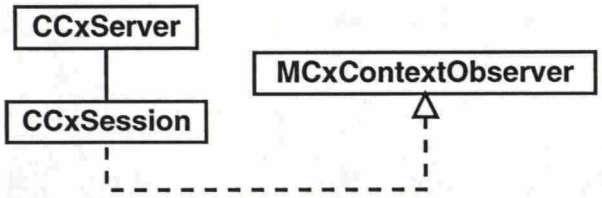

Figure 4.10: UML diagram of the Core of the Context Server

tion to use all services of the Context Server. The library consists of one class only, so no class diagram is presented here. The methods are described in Table 4.6.

Method	Description
Connect	Connects to the server
Close	Disconnects from the server
Version	Returns the version of the server
Connected	Returns true, if the server is connected
NumContexts	Returns the number of context names
ContextNamesL	Returns a list of context names
SetContextL	Sets a value to a context
ContextL	Returns the value of a context
SubscribeL	Notifies the server of the context types the client is interested in
WaitNotification	Requests a notification if one of the context types subscribed to is changed
WaitNotificationCancel	Cancels the WaitNotification call
Shutdown	Shuts the server down. For debugging only

Table 4.6: Methods provided by class `RCx`

4.2.4 Plugins

The plugins are dynamic link libraries. Each dynamic link library contains one plugin. The loading mechanism is described in Chapter 4.2.2.

In theory, the plugins could have been implemented to use the Context Client interface, because the interface provides all the services that the plugins need to access. The problem with the Context Client interface is that every service request would

go via the Symbian OS message passing mechanism. It includes context switches, which can be a major bottleneck in the performance-critical mobile devices. The plugin mechanism provides an optimized way of using the services.

DrDAQ

The DrDAQ plugin retrieves information from the DrDAQ data logger. The data is transferred to the mobile device via a Bluetooth connection, and the plugin processes the data. It calculates the average, minimum, maximum and zero-crossing rate for each sensor type and packs the data into data packages. These packages are stored into `CCxContextBoard`. For testing purposes, a variant of this plugin was made, which retrieves the sensor data from a specially named file.

The performance of the plugin was also evaluated. The results are in Chapter 4.3.1 for the Bluetooth transmission and Chapter 4.3.2 for calculating the statistics.

NetInfo

The NetInfo plugin connects to the Telephony Server of Symbian OS. The plugin registers itself as an observer for network information changes. The Telephony Server notifies the plugin every time the mobile device switches to another GSM cell. The plugin maps the information into a location described as a string, if possible, and feeds the raw network information and interpreted location into `CCxContextBoard`. The plugin also registers itself as an observer to the `CCxContextBoard`. This was done so that the module can be taught new mappings for locations. All changes done by any other module than the NetInfo plugin are stored into the mapping table of the plugin.

The NetInfo plugin was not tested formally. Instead, it was tested in real life conditions. The plugin was run on the mobile device, while the device was transferred to different locations in Helsinki and Espoo. Some of the locations visited were taught to the plugin by assigning names for them. After a while, the plugin knew most of the locations retrieved from the Telephony Server. The positioning data retrieved is very rough, but it was used to determine the nearest bus stop to the user. For that, it performed well. There were some problems, when two locations were close to each other, and they had an overlapping cell.

The most severe problem with the plugin was that the user could not choose the current location from a list of known locations. Instead, he had to enter the name

of the location every time. Fortunately, the teaching period has to be done only once.

Calendar

The Calendar plugin registers itself as an observer to the Agenda Server of Symbian OS. Each time the calendar is modified, the plugin is notified by the Agenda Server. The plugin then reads the information for the next appointment stored in calendar. It stores the location field of the next appointment as well as the starting time of the appointment in `CCxContextBoard`.

No formal tests were performed for the Calendar plugin, either. It was used in real-life situations with the bus schedule application. Its performance was solid.

The greatest problem with this plugin was the same as the problem with the NetInfo plugin — the user must enter the location of the meeting to his calendar in exactly the same way, as the bus schedule application knows them. Only capitalization is allowed to change.

Kohonen's Self-Organizing Map

The Kohonen's Self-Organizing Map plugin registers itself as an observer for the information retrieved by the DrDAQ plugin. Whenever the DrDAQ plugin has written new information for all sensors, the KSOM plugin reads the information listed in Table 3.3. The information is formatted as a vector and entered into the k-means augmented Kohonen's self-organizing map as described in 3.2.4. The network determines a scalar context number. This number is stored into the `CCxContextBoard`. Whenever this information is changed by some other module than the KSOM plugin, it is fed to the network as training data.

The plugin was tested with sensor data. The results are in Chapter 4.3.3 and Chapter 4.3.4 for performance testing and Chapter 4.3.6 for testing the accuracy of the output of the plugin.

4.3 Performance of the Context Server

The Context Server is designed to run in a mobile device with little resources, so it must be efficient. The algorithms must be fast enough to allow normal operation of the mobile device. In addition, the battery consumption depends on the number of operations performed, so this has to be considered, too.

The performance was measured for each algorithm with varying sets of data. It was measured in a real-life like situation, where all the plugins were loaded to the server, and in addition one client application was connected and subscribed to all context types.

For the self-organizing map and k-means clustering, also the accuracy of the results was considered.

4.3.1 Transmission of Context Data

The performance of transmission was measured by sending data packets of various sizes to the mobile device. The numbers of data samples in the packets is listed in Table 4.7. The number of data samples was constant for all other sensor types but the sound waveform. This does not affect the results, because the transmission time depends only on the number of bytes transmitted. The size of the packets in bytes can be calculated with the formula

$$size = numSamples \cdot 2 + 4 \cdot 2 + 2. \tag{4.1}$$

Each data sample is 16 bits wide, so they take two bytes. Before each sensor type, the number of samples is sent as a 16-bit integer. In addition to this, the whole packet is prepended with the total number of bytes sent as a 16-bit integer.

The transmission was repeated 290–651 times, and the average time was used. Because the system clock ticks 64 times per second, the times are rounded to the nearest 64th of a second. During the tests there were no other Bluetooth transmissions going on near the scene. The sending and receiving devices were located less than 10 centimeters apart.

The transmission times are listed in Table 4.8.

Sound waveform	Sound level	Temperature	Light	Total
1000	100	100	100	1300
5000	100	100	100	5300
10000	100	100	100	10300
20000	100	100	100	20300

Table 4.7: Sizes of data packets sent via Bluetooth in number of data samples

Total samples	Repetitions	Average time	Minimum	Maximum
1300	290	567	469	1703
5300	516	1976	1844	2313
10300	651	3756	3641	8828
20300	291	7339	3953	7484

Table 4.8: Transmission times of data packets sent via Bluetooth in milliseconds

4.3.2 Calculating Statistics from Context Data

The processing time for statistics was measured with the test data described in Table 4.7. Because the processing time for the statistics only depends on the total number of samples, only the number of samples of the sound waveform was varied in the tests.

Total samples	Repetitions	Average time	Minimum	Maximum
1300	289	11	0	16
5300	516	39	31	47
10300	651	69	63	94
20300	290	128	109	141

Table 4.9: Time to calculate statistics of sensor data in milliseconds

4.3.3 Feeding the Statistics to Self-Organizing Map

In this test, the number of samples in the test data does not affect the results. This is because only the statistics of the sensor data are used. They contain the same amount of data irrespective of the number of data points they represent.

The time was calculated from the point just before feeding the statistics to the self-

organizing map. It ended right after the network returns with a winning neuron. The times are listed in Table 4.10.

Neurons	Repetitions	Average time	Minimum	Maximum
100	461	12	0	16
196	250	14	0	31
400	218	17	0	47

Table 4.10: Time for the self-organizing map to process data in milliseconds

4.3.4 Retrieving a Label from K-Means Clustering

As in the performance calculation of self-organizing map, this test is independent of the number of samples. On the other hand, it is very much dependent on the number of cluster sets. Each cluster set represents a single context type.

The test was done on a cluster set, which had already been fully initialized. This means that all the cluster sets had been filled with random vectors. The times are listed in Table 4.11.

Number of cluster sets	Repetitions	Average time	Minimum	Maximum
10	215	14	0	31
20	568	13	0	31
40	432	13	0	31

Table 4.11: Time for the k-means algorithm to process data in milliseconds

4.3.5 Total Processing Times

The tests of individual algorithms ignore the time it takes for the data to propagate from one plugin to another. When the data is received from the network, its statistics are entered into the Context Engine. After that, the self-organizing map is notified of the new data. After the self-organizing map and k-means algorithm have processed the data, the results are entered into the Context Engine.

The self-organizing map contained 196 neurons. That number was chosen, because it gave good performance, but is also memory efficient. The k-means algorithm was initialized with 10 full cluster sets. That corresponds to 10 different

context types assigned by the user. The number of data samples was varied according to Table 4.7.

This test measures the total time it takes to process a single packet of samples. The time starts when the Bluetooth connection is established between the mobile device and the desktop computer. The time ends when the results of k-means algorithm have been sent to the Context Engine. It includes the overhead of routing the sensor statistics via the Context Engine, but also the overhead of the self-organizing map and the k-means algorithm saving their internal state to flash memory. The durations are listed in Table 4.12.

Because the execution order of the individual operations is not defined, the times were slightly contaminated. The iterations, where the transfer of the next block of data was interleaved with processing the previous block of data, were excluded from the statistics.

Total samples	Repetitions	Average duration	Maximum	Minimum
1300	243	805	969	750
5300	214	2303	2031	3063
10300	91	4240	5468	3906
20300	271	7686	7656	8859

Table 4.12: Total processing time in milliseconds

The relative times used by individual operations were measured also. The Figure 4.11 shows a breakdown of the times used by different operations in total time.

4.3.6 Correctness of Determining Context

Due to the inability of the measuring equipment to work on the laptop computer, the measurements had to be done in office environment, as described in Chapter 3.3.

The following context types were used: office at daytime, office at night, office at day with the measuring equipment located near the user and office at day with the measuring equipment hanging from the window. The context types were named Day, Night, Belt, and Out respectively. The transition from one context type to another was included in the measurements, so the samples describe one possible day for the Context Server. The measurements were taken with an interval of six

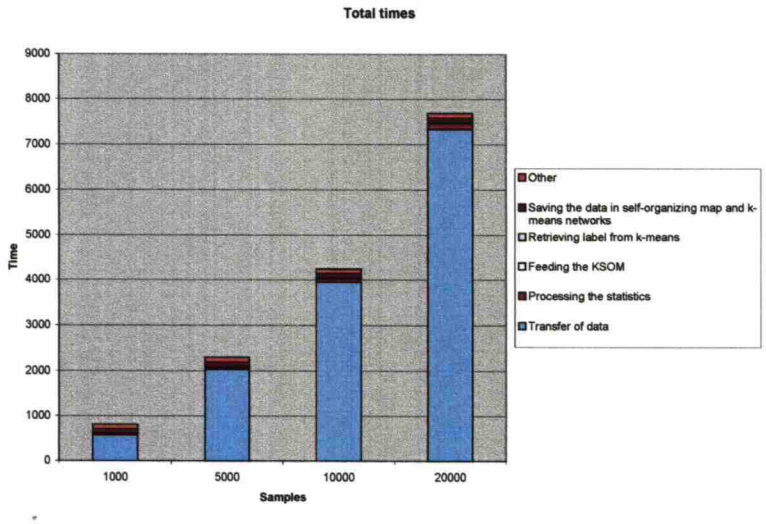

Figure 4.11: Breakdown of operations in total processing time

seconds and the total number of samples was 14321. The script that describes the context types during the measurement period is in Table 4.13.

Time	Action
16:04	Start of measurements
18:30	Approximate start of nighttime context
08:30	Approximate start of daytime context
10:12	Device put near the user
10:35	Device put back on table
10:38	Device put back near the user
11:16	Device put back on table
13:44	Device put hanging outside the window
14:44	Device put back on table
16:24	Measuring ended

Table 4.13: Script of the measurements of test data

A driver application was created to enable automation of the testing. At first, the system was trained with labeled samples. The samples were entered in the same order as they were measured, and approximately 1% of the samples were labeled.

In the actual test, the samples were entered in the same way, as in the training period. For each sample, the label the Context Server suggested was recorded and compared to the correct label. The number of correct labels versus incorrect labels

was calculated for each minute of measurement data. The accuracy is depicted in Figure 4.12. The time is in horizontal axis and the accuracy in vertical axis. The name of the current context can be found on top of the graph. When the accuracy is 1.0, Context Server was able to determine the correct context perfectly at that point of time.

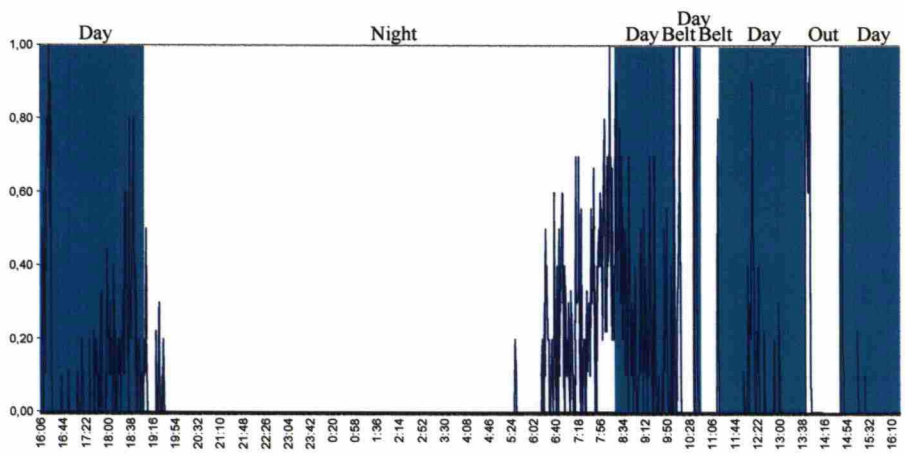

Figure 4.12: Accuracy of the detection of the current context

4.4 Example Application

An example application was created, which uses the Context Server to provide the user with a richer experience. The goal was for the application to perform a truly useful task, but not necessarily to use all the information that the Context Server can provide. The application was divided to different parts called views to demonstrate the implementation of all categories of context-awareness. The categories are described in Chapter 1.2.

The bus schedule view contains a partial list of busses from the Helsinki area in a database. When started, it reads the current location and the location from the Calendar and lists the busses that take the user to the intended place. The database part was modified from a previously written application for the Psion Revo palm top. It demonstrates presentation of information and service to the user.

The reminder view allows the user to enter reminders and attach them to locations. When the user passes the location, the application shows the reminder to the user. It demonstrates automatic execution of service and tagging information for later retrieval.

The application uses only the simpler types of context information that the Context Server provides.

4.4.1 Structure of Application

The application consists of three parts: the application, its views and the model. It is designed using a slightly modified version of the design pattern model-view-controller common for all Symbian OS applications[14].

In model-view-controller, the application is divided into three distinct parts[14]. The model contains the data of the application, and algorithms for manipulating the data. The view shows the data in the model to the user. The controller updates the model and requests redraws from the view.

In Symbian OS, views refer to what the user sees as the user interface. It includes the MVC view, but also the MVC controller for that particular part of the application. This can create confusion, because the term view is used in two different meanings, so the views in the model-view-controller design pattern are referred MVC views.

Application

The application creates the views and the model. It handles switching between the different views, and exiting. The class diagram is in Figure 4.13.

The classes `CCxUiApp` and `CCxUiDocument` are classes required by the Symbian OS framework. They contain no further semantics in this application. The class `CCxUiAppUi` is the controller of the user interface, and creates all user interface elements. The application contains three views, which can be switched by the user.

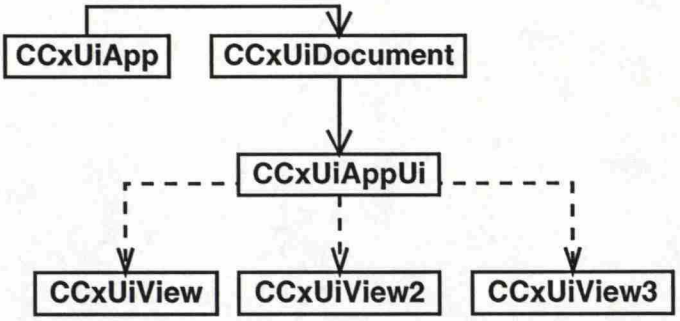

Figure 4.13: Class diagram of the bus schedule application

Views

The views convert the information received from the models into a human readable form. They also contain the application logic. Each view presents a different feature to the user, so they can partially be considered controllers. There are three views. The class diagrams are in Figure 4.14, Figure 4.15 and Figure 4.16. The classes `CCxUiView`, `CCxUiView2` and `CCxUiView3` are the controller classes for the views. The classes `CCxUiContainer`, `CCxUiContainer2` and `CCxUiContainer3` contain the MVC views.

The first view, whose controller is in class `CCxUiView` and MVC view in `CCxUiContainer`, is only for debugging purposes. It shows all the context information retrieved from the Context Server. It has an observer, `CCxUiContextObserver`, which has been registered to the Context Server. Whenever changes occur in the context information, the observer class requests a redraw.

The second view, whose controller is in class `CCxUiView2` and MVC view in `CCxUiContainer2`, displays the bus schedules to the user. It also contains links to the bus schedule via `CBusEng`.

The third view, whose controller is in class `CCxUiView3` and MVC view in `CCxUiContainer3`, is a user interface for the reminder feature. The user can attach a note to a certain location. When the user passes the location, he is shown the note.

The views each create their own connections to the server. This makes it easy to separate them into standalone applications, if needed.

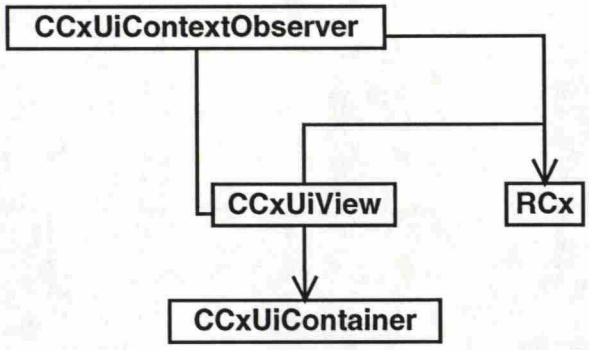

Figure 4.14: Class diagram of the bus schedule view 1

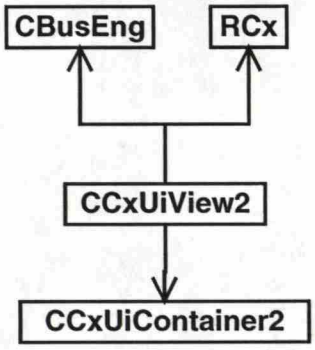

Figure 4.15: Class diagram of the bus schedule view 2

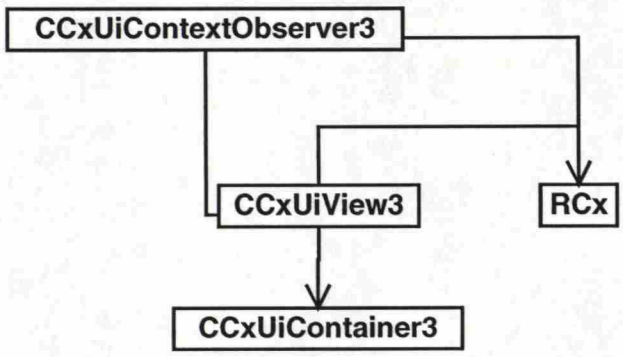

Figure 4.16: Class diagram of the bus schedule view 3

Model

The model uses the Symbian OS database manager, which offers limited relational database functionality[14], to store the bus schedules. The model also offers incremental loading of new schedules, which allows the application to respond to user input even while loading new data.

The model stores information on the routes of individual bus lines. It also stores information on the times when the busses are in each bus stop. This information is originally downloaded from the YTV web site `http://www.ytv.fi` and processed before it is entered into the bus schedule application.

The class `CBusDB` takes care of the database. It contains methods for retrieving and searching for information. The class `CBusEng` uses the database class for information retrieval, and offers higher-level methods to its clients. The class diagram is in Figure 4.17.

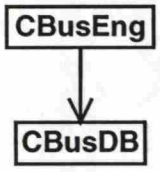

Figure 4.17: Class diagram of the bus schedule model

Chapter 5

Analysis

5.1 Context Server System

The architecture of the Context Server serves its purpose well. The plugins were easy to write, and the performance of the Context Server was satisfactory. The Context Server satisfies all the use cases specified in Chapter 4.1, and all requirements specified in Chapter 2.1.

The architecture proved to be sound when the plugins and the example application were written. The simpler plugins were written in a few days, and the application described in Chapter 4.4 was adapted from an old non-context aware design in two days. This shows that the architecture really supports writing context aware applications.

The architecture lacks a layer, which enforces the format of the context values. For example, the locations entered into the server were not enumerated anywhere. Thus, the user could enter locations with small spelling mistakes, and there would be no safeguard against it. The architecture should have a way of enumerating the choices for a certain context type so that the applications could show a list of choices for the user. In addition, the applications can enter the data in a completely wrong format. For example, the application could send an integer for the context type where a string was expected. These safeguards could be implemented as a layer on top of the Context Client interface. This layer should also help the application to interpret the data.

A new plugin could be developed, which launches applications according to certain conditions in the user's environment. It could, for example, act as a reminder

for a shopping list when the user walks by a grocery store.

5.2 Context Server Plugins

The plugins that created the most interesting information were also the simplest plugins: Calendar and NetInfo plugins. Their information was used for developing an application with real uses.

The DrDAQ plugin provides low-level information about the environment of the user. Alone its uses are very limited, but when coupled with another plugin to interpret the information, it could be useful.

The KSOM plugin derives a context from the information provided by the DrDAQ plugin. Deriving the context worked adequately, but the usefulness of the derived context could not be established.

5.3 Performance of the Context Server

The results in Table 4.8 show that the transmission time of the sensor data is noticeable. The time increases linearly, which suggests that there is a bottleneck in the transmission, because Bluetooth can theoretically send one megabits per second. The serial interface of the Ericsson Bluetooth Application Toolkit is probably the reason. Differences can also be caused by the asynchronous nature of the application programming interface. The transmission of the data package may have been completed while the Context Server was performing other operations, so the plugin was notified later. The differences between minimum and maximum times could be attributed to this delay.

As can be seen in the results in Table 4.9, the time used in calculating the statistics increases nearly linearly with the total number of samples. In every case, the time is very small, and the minimum and maximum times are relatively close to each other. The minimum time of zero is due to the limited accuracy of the system clock.

From the results in Table 4.10, it is obvious that for a 10×10 self-organizing map, the time to feed the network with a new value is negligible. In the results, the accuracy of the system clock is again the limit. This also happens for maps of size 20×20, so the time spent in self-organizing map algorithm contributes very

little to the overall performance of the system.

The k-means algorithm was also tested and the results were presented in Table 4.11. The results show that the algorithm is fast enough not to produce measurable results when timing the operations.

In all, the performance of the Context Server is very good. The results in Table 4.12 and Figure 4.11 show that most of the time is spent transferring the data. If the bottleneck of the Bluetooth transmission is not considered, the algorithms work in near real-time. The battery consumption was tested by leaving the Context Server running for long periods. The battery consumption was not increased noticeably.

5.3.1 Deriving Context Information

In general, the results of the test show that the accuracy is at times excellent, but at times very low. The error density is the highest in the transition between two contexts, which was expected. The number of errors fell quickly after the short transition periods to almost zero.

It seems that the labeling was done incorrectly, because most errors were generated between 18:00 and 19:30 of the first day and between 6:00 and 8:30 of the second day. Presumably, people were working in the office during that time and thus the measurement data was too much like the measurement data of daytime.

The error density varies slightly, but in general, it seems to be on a satisfactory level. The problems were in the transition periods, but they did not last for long. Also small spikes can be seen every now and then. They are probably due to temporary changes in the environment. They could probably have been avoided by using the Markov chains suggested in [17].

The final reason for problems could be the failure to take measurements in real-life conditions. All the measurements were taken in an office environment, and it can be that the different contexts chosen were not different enough to be noticeable with the chosen sensors.

5.4 Example Application

The application was used in real-life activities. Its performance was very good, and it did not consume much energy. Often the application was left running for hours while the mobile device was not in use. The application as such was not very useful, and several ideas of improvement came up.

The bus schedule application depends only on the Context Server to provide the location data. This means that the user must have a calendar entry for the target location every time he wants to view the bus schedule. The application should allow the user to choose the values, if he so desires.

The reminder application was used for reminding about buying something from the grocery store. The application shows a note accompanied with a short sound when there is a remainder. This can easily be left unnoticed, so the application needs to use the vibrating alert and a longer sound sequence to notify the user.

In all, the application did gain much from the automatic usage of the user's context. This is remarkable, since the context information used in the application was only the location and calendar of the user.

Chapter 6

Conclusions

The research problem was defined in the introduction with four questions.

How can the context information be acquired? This research has shown the devices and methods for acquiring information from the user's context.

What kind of contextual information is a mobile device able to use? Example uses for information have been presented, but this question needs further study. There is a very large amount of information that can be used and this research has merely scratched the surface.

What kinds of sensors are needed to produce meaningful information? The sensors in the DrDAQ data logger provide a good basis for the sensors needed. Even the small number of sensors in the board can be used for separating different contexts when they are used wisely. This research did not try to study a wider range of sensors. Most likely adding other sensors would yield even more information about the user's context. It was also shown that meaningful information could be provided just by using the information already available on a mobile device with no extra devices.

How could applications take advantage of the context information? The Context Server shows that a component with plugin capabilities and a simple interface allows applications to be made context-aware in a very short time. The architecture also allows adding new context information to the system by using plugins.

In addition to the research questions, four research goals were set in the beginning of this research. As these goals were considered the most important outputs from this research, they are also good issues with which to conclude this research.

The first goal was to determine what kind of sensor information a mobile device is able to process to produce meaningful information about its context. A combination of a self-organizing map and k-means clustering was assessed. The combination was fast enough to run on a mobile device, and its accuracy was as expected. In addition, meaningful information was extracted from the mobile device itself. Namely, the location of the device, the current time and the user's calendar were used to provide information about the user's intent.

The second goal was to create a system, Context Server, which enables applications to take advantage of the context easily. This goal was reached, and the interface proved to be easy in converting a legacy application to be context-aware. The server was designed so that different types of context plugins can be added to the system. The implementation of the plugins is simple.

The third goal was to produce a small example application, which is able to take advantage of the contextual information created by Context Server. A legacy bus schedule application was converted to be context-aware, and it was augmented with a reminder application. These together demonstrate that all three categories of context-awareness can be implemented in applications with the help of the Context Server.

The last goal was to develop a demonstration prototype, which consists of sensors, media to transfer the sensor data to a mobile device, and the mobile device. This was created, but the hardware proved to be problematic. The data logger works only on the desktop computer used, and failed to work on a laptop. The system works also only with the mobile device. In that case, only a subset of the context information is available to the applications.

All goals were met partially or fully. The prototype system built was not as close to a real system as planned due to the failure of the measuring equipment, but the three other goals were met fully.

Bibliography

[1] Bluetooth application tool kit. http://www.ericsson.com/bluetooth/products&s/show_product.asp?show=deliverables&cat=49&prodid=58, March 2002.

[2] G. Booch, J. Rumbaugh, and Jacobson I. *The Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, 1999.

[3] A.K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf, 2000.

[4] A.K. Dey and G.D. Abowd. Toward a better undestanding of context and context-awareness. GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999.

[5] Drdaq data logger from pico technology. http://www.drdaq.com/, March 2002.

[6] D. Gerhard. *Audio Signal Classification*. PhD thesis, School of Computing Science, Simon Fraser University, 2000.

[7] S. Haykin. *Neural Networks A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1999.

[8] S. Kaski. Data exploration using self-organizing maps. In *Acta Polytechnica Scandinavica, Mathematics*, number 82 in Computing and Management in Engineering Series. Finnish Academy of Technology, 1997.

[9] T. Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer-Verlag, 1984.

[10] T. Kohonen. *Self-organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer-Verlag Heidelberg, 1995.

[11] Nokia. *Nokia Coding Conventions for Symbian OS*, 2002. Nokia Series 60 SDK for Symbian OS.

[12] Nokia 7650. `http://www.nokia.com/phones/7650/`, March 2002.

[13] A. Schmidt, K.A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, 1999.

[14] M. Tasker. *Professional Symbian Programming*. Wrox Press, 2000.

[15] The Symbian Platform. Nokia 9210 SDK, can be ordered for free from `http://www.forum.nokia.com`, 2001.

[16] The Symbian Platform White Paper. `http://www.forum.nokia.com`, under section Symbian, April 2002.

[17] K. van Laerhoven. Combining the self-organizing map and k-means clustering for on-line classification of sensor data. In *Proceedings of the International Conference on Artificial Neural Networks*. ICANN, 2001.